

Experiment 5

Name: Saksham

Branch: BE-IT

Semester: 6th

Subject: Project Based Learning in Java

UID: 22BET10003

Section: IOT703/A

Date of Performance: 18/02/25

Subject Code: 22ITH-359

Problem 1:

1. Aim: Write a Java program to calculate the sum of a list of integers using autoboxing and unboxing. Include methods to parse strings into their respective wrapper classes (e.g., Integer.parseInt()).

2. Code:

```
package exp;
import java.util.*;

public class Exp5_1 {
    public static int calculateSum(List<Integer> numbers) {
        return numbers.stream().mapToInt(Integer::intValue).sum(); // Efficient sum calculation
    }

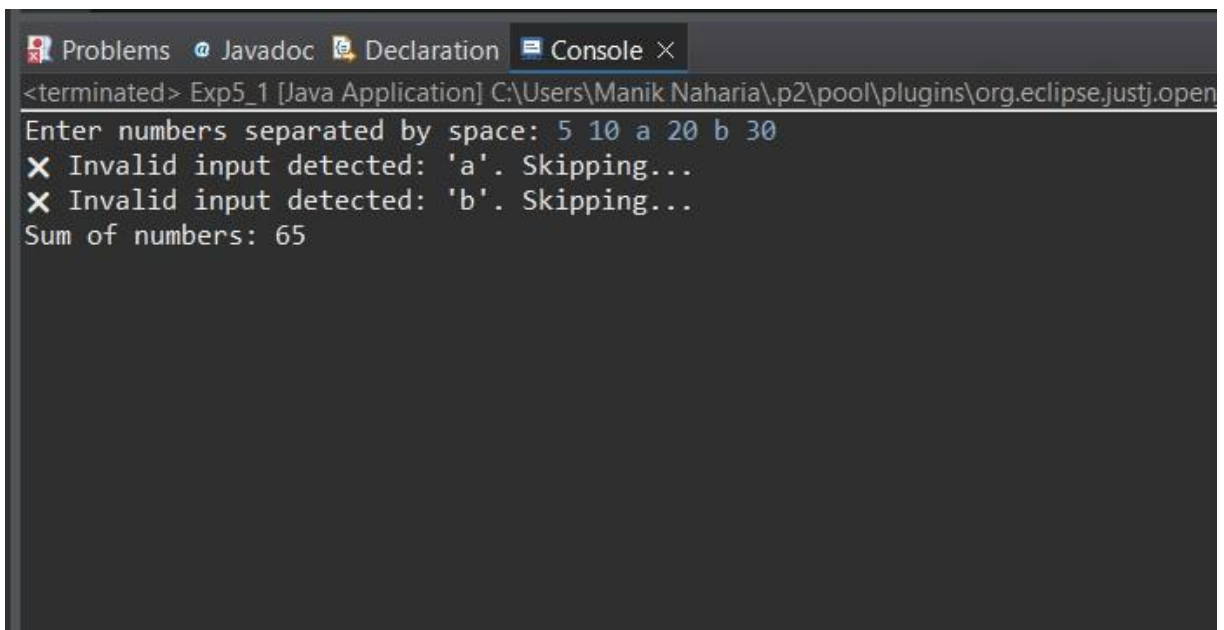
    public static List<Integer> parseStringToIntegers(List<String> strNumbers) {
        List<Integer> numbers = new ArrayList<>();
        for (String str : strNumbers) {
            try {
                numbers.add(Integer.parseInt(str)); // Convert string to integer
            } catch (NumberFormatException e) {
                System.out.println(" Invalid input detected: '" + str + "'. Skipping...");
            }
        }
        return numbers;
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter numbers separated by space: ");
        String input = scanner.nextLine();
        scanner.close();

        List<String> strNumbers = Arrays.asList(input.trim().split("\\s+")); // Trim & handle multiple spaces
        List<Integer> numbers = parseStringToIntegers(strNumbers);
    }
}
```

```
        if (numbers.isEmpty()) {  
            System.out.println("No valid numbers entered.");  
        } else {  
            System.out.println("Sum of numbers: " + calculateSum(numbers));  
        }  
    }  
}
```

3. Output:



The screenshot shows the Eclipse IDE's Console window. The title bar includes 'Problems', 'Javadoc', 'Declaration', and 'Console'. The console text shows the program's execution: it prompts for numbers, receives '5 10 a 20 b 30', skips the invalid inputs 'a' and 'b', and finally prints the sum of the valid numbers as 65.

```
<terminated> Exp5_1 [Java Application] C:\Users\Manik Naharia\p2\pool\plugins\org.eclipse.justj.open  
Enter numbers separated by space: 5 10 a 20 b 30  
X Invalid input detected: 'a'. Skipping...  
X Invalid input detected: 'b'. Skipping...  
Sum of numbers: 65
```

4. Learning Outcomes:

- Understand autoboxing and unboxing between primitives and wrapper classes.
- Learn to parse strings into integers using `Integer.parseInt()`.
- Utilize Java collections to store wrapper objects.
- Iterate over collections with enhanced for-loops.

Problem 2:

- 1. Aim:** Create a Java program to serialize and deserialize a Student object. The program should:
Serialize a Student object (containing id, name, and GPA) and save it to a file.
Deserialize the object from the file and display the student details.
Handle FileNotFoundException, IOException, and ClassNotFoundException using exception handling.

2. Code:

```
import package exp;

import java.io.*;

class Student implements Serializable {
    private static final long serialVersionUID = 1L;
    private int id;
    private String name;
    private double gpa;

    public Student(int id, String name, double gpa) {
        this.id = id;
        this.name = name;
        this.gpa = gpa;
    }

    public void display() {
        System.out.println("Student ID: " + id);
        System.out.println("Student Name: " + name);
        System.out.println("Student GPA: " + gpa);
    }
}

public class Exp5_2 {
    private static final String FILE_NAME = "student.ser";

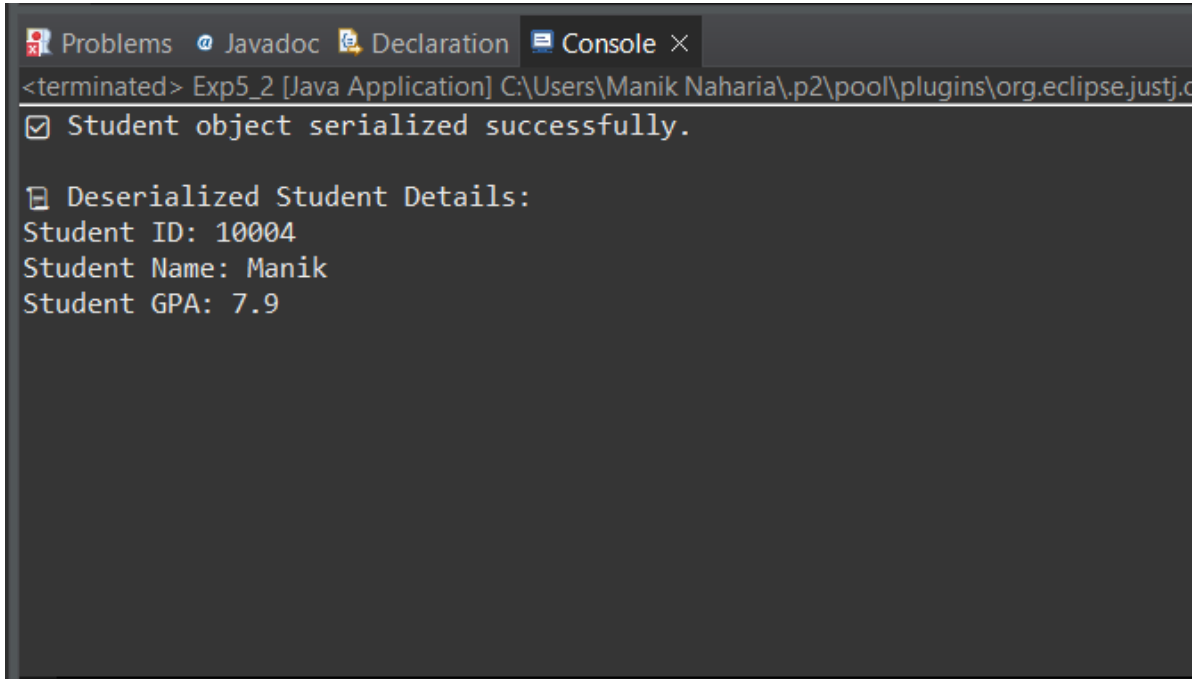
    public static void serializeStudent(Student student) {
        try (ObjectOutputStream oos = new ObjectOutputStream(new
        FileOutputStream(FILE_NAME))) {
            oos.writeObject(student);
            System.out.println("Student object serialized successfully.");
        } catch (FileNotFoundException e) {
            System.out.println("Error: File not found. " + e.getMessage());
        } catch (IOException e) {
            System.out.println("Error: Unable to serialize object. " + e.getMessage());
        }
    }

    public static void deserializeStudent() {
        try (ObjectInputStream ois = new ObjectInputStream(new
        FileInputStream(FILE_NAME))) {
```

```
        Student student = (Student) ois.readObject();
        System.out.println("\n Deserialized Student Details:");
        student.display();
    } catch (FileNotFoundException e) {
        System.out.println("Error: File not found. Run serialization first. " +
e.getMessage());
    } catch (IOException e) {
        System.out.println("Error: Unable to deserialize object. " + e.getMessage());
    } catch (ClassNotFoundException e) {
        System.out.println("Error: Student class not found. " + e.getMessage());
    }
}

public static void main(String[] args) {
    Student student = new Student(10004, "Manik", 7.9);
    serializeStudent(student);
    deserializeStudent();
}
}
```

3. Output:



```
<terminated> Exp5_2 [Java Application] C:\Users\Manik Naharia\.p2\pool\plugins\org.eclipse.justj.c
☑ Student object serialized successfully.

☐ Deserialized Student Details:
Student ID: 10004
Student Name: Manik
Student GPA: 7.9
```

4. Learning Outcomes:

- Understanding Java serialization/deserialization.
- Implement the Serializable interface.
- Use try-with-resources for stream management.
- Handle exceptions: FileNotFoundException, IOException, ClassNotFoundException.

Problem 3:

- 1. Aim:** Create a menu-based Java application with the following options. 1. Add an Employee
2. Display All 3. Exit If option 1 is selected, the application should gather details of the employee like employee name, employee id, designation and salary and store it in a file. If option 2 is selected, the application should display all the employee details. If option 3 is selected the application should exit.

2. Code:

```
package exp;
import java.io.*;
import java.util.*;

class Employee1 implements Serializable {
    private static final long serialVersionUID = 1L;
    private int id;
    private String name;
    private String designation;
    private double salary;

    public Employee1(int id, String name, String designation, double salary) {
        this.id = id;
        this.name = name;
        this.designation = designation;
        this.salary = salary;
    }

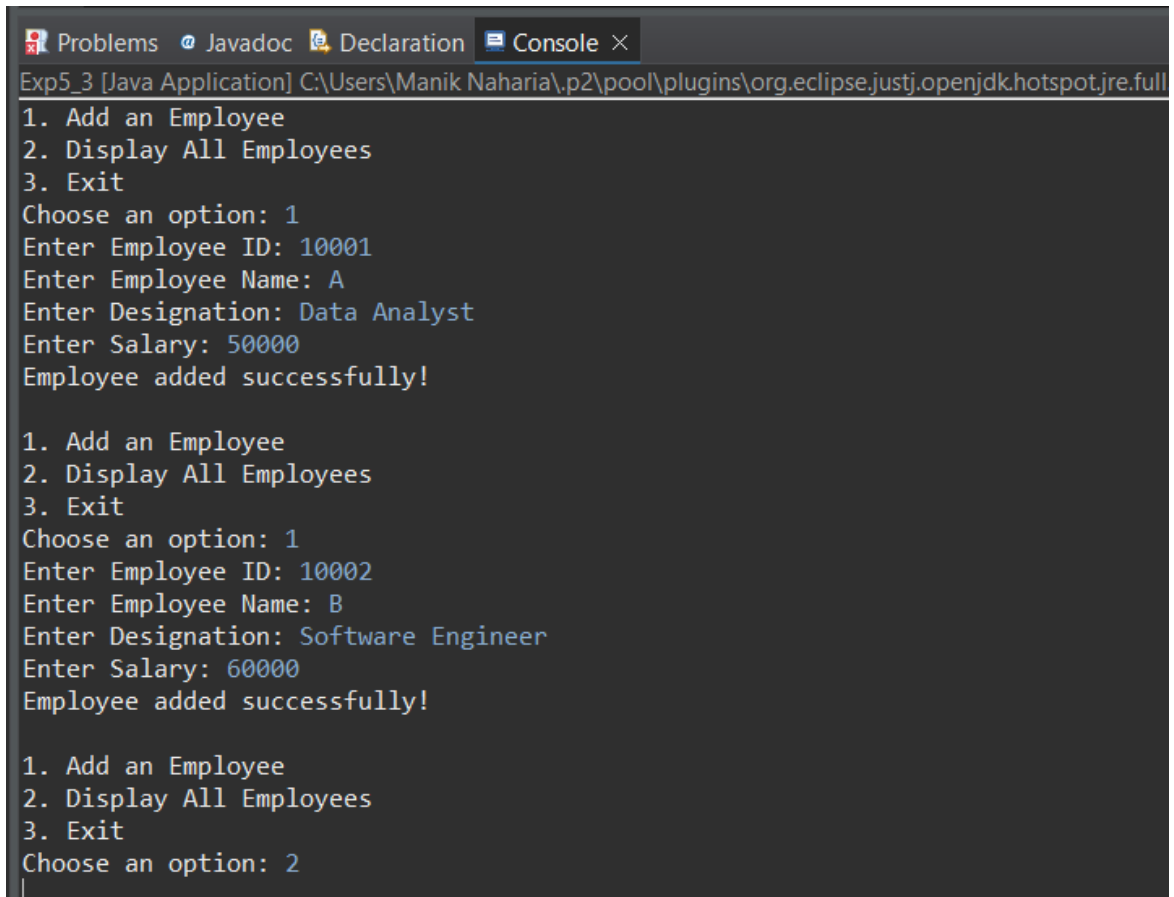
    @Override
    public String toString() {
        return "ID: " + id + ", Name: " + name + ", Designation: " + designation + ", Salary: " + salary;
    }
}

public class Exp5_3 {
    private static final String FILE_NAME = "employees.dat";
    private static Scanner scanner = new Scanner(System.in);

    public static void addEmployee() {
        System.out.print("Enter Employee ID: ");
        int id = scanner.nextInt();
        scanner.nextLine(); // Consume newline
        System.out.print("Enter Employee Name: ");
        String name = scanner.nextLine();
        System.out.print("Enter Designation: ");
        String designation = scanner.nextLine();
        System.out.print("Enter Salary: ");
        double salary = scanner.nextDouble();
    }
}
```

```
Employee1 employee = new Employee1(id, name, designation, salary);
saveEmployeeToFile(employee);
System.out.println("Employee added successfully!\n");
}
public static void saveEmployeeToFile(Employee1 employee) {
    try (ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream(FILE_NAME, true))) {
        oos.writeObject(employee);
    } catch (IOException e) {
        System.out.println("Error: Unable to save employee.");
    }
}
public static void displayAllEmployees() {
    try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(FILE_NAME))) {
        System.out.println("\nEmployee Details:");
        while (true) {
            Employee1 emp = (Employee1) ois.readObject();
            System.out.println(emp);
        }
    } catch (EOFException e) {
        // End of file reached
    } catch (FileNotFoundException e) {
        System.out.println("No employee records found.\n");
    } catch (IOException | ClassNotFoundException e) {
        System.out.println("Error reading employee records.\n");
    }
}
public static void main(String[] args) {
    while (true) {
        System.out.println("1. Add an Employee");
        System.out.println("2. Display All Employees");
        System.out.println("3. Exit");
        System.out.print("Choose an option: ");
        int choice = scanner.nextInt();
        switch (choice) {
            case 1:
                addEmployee();
                break;
            case 2:
                displayAllEmployees();
                break;
            case 3:
                System.out.println("Exiting...");
                return;
            default:
                System.out.println("Invalid choice! Try again.\n");
        }
    }
}
```

3. Output:



```
Problems Javadoc Declaration Console ×
Exp5_3 [Java Application] C:\Users\Manik Naharia\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full
1. Add an Employee
2. Display All Employees
3. Exit
Choose an option: 1
Enter Employee ID: 10001
Enter Employee Name: A
Enter Designation: Data Analyst
Enter Salary: 50000
Employee added successfully!

1. Add an Employee
2. Display All Employees
3. Exit
Choose an option: 1
Enter Employee ID: 10002
Enter Employee Name: B
Enter Designation: Software Engineer
Enter Salary: 60000
Employee added successfully!

1. Add an Employee
2. Display All Employees
3. Exit
Choose an option: 2
```

4. Learning Outcomes:

- Learn to create a menu-driven console application.
- Understand file I/O for reading and writing text files.
- Implement exception handling for file operations.
- Practice gathering and processing user input using Scanner.